

Data Request Support

Acnet RETDAT protocol

Tue, Oct 29, 2002

This note describes software support for the Acnet RETDAT protocol as implemented in the MC68040-based IRM and Linac PowerPC-based front-ends.

Acnet RETDAT protocol

The generic Acnet data request protocol is called RETDAT. Its Acnet task name is included in the 9-word Acnet header, described later. The message body consists of three 16-bit words followed by an array of 8-word structures, each of which refers to an Acnet device. The initial fields are as follows:

<i>Field</i>	<i>Size</i>	<i>Meaning</i>
nBTotal	2	Total #bytes expected in body of reply message
nDev	2	#device packets
ftd	2	Frequency-time descriptor

The format of the 8-word structure that describes each device packet is as follows:

<i>Field</i>	<i>Size</i>	<i>Meaning</i>
pidiLo	2	Device index low 16 bits
pidiHi	2	Property index in hi byte, device index upper 8 bits in lo byte
ssdn	8	Subsystem Device Number
length	2	#bytes requested
offset	2	Offset specified

The nBTotal announces the total size of the expected reply data, including the status word that precedes each device packet reply data. The nDev is the count of device packets included in the request. The ftd has three forms that specify when reply data is to be returned:

0000	One-shot request. Return data one time, right away.
0ppp	Periodic, where ppp is the period in units of 60 Hz cycles.
80xx	Clock event, where xx is the Tevatron clock event#

The Acnet header that precedes any Acnet message body has the following format:

<i>Field</i>	<i>Size</i>	<i>Meaning</i>
msgType	2	Message type as listed above
status	2	Status word used by reply messages
destNode	2	Destination node#
srcNode	2	Source node#
taskName	4	Task name
srcTskId	2	Source task Id
msgId	2	Message id
msgSize	2	Message size, including 18-byte Acnet header

The first word of the Acnet header specifies the message type in the low 4 bits as follows:

0000	USM (unsolicited message). No reply.
0002	One-shot request. Only one reply expected.
0003	Request specifying multiple replies. May be periodic or clock-event.
0004	Reply to one-shot request
0005	Reply to request specifying multiple replies
0200	Cancel request. Used to terminate multiple replies.

The other bits in the msgType word may have uses, but only those listed are supported.

The destNode and srcNode fields specify Acnet node#.s. After swapping bytes in each field, each resulting hi byte is sometimes called the lan#, and the low byte is called the node#. In this note, it is merely called an Acnet node#. Note that the node# fields are always from the point of view of the requesting node. The Acnet header of a reply message to an Acnet request is easily formed by first

copying the Acnet header of the request message, then modifying the `msgType` to 0004 or 0005, setting the status word value and the appropriate message length for the reply.

The `taskName` is in RAD-50 encoded format, in which 3 characters are compressed into 16 bits by means of a base-40 notation. (This works because 40 cubed is less than 2^{16} . Used heavily in the DEC PDP-11 days, it is called RAD-50 because 50 is the octal representation of 40 decimal.) So the 6-character name `RETDAT` can be squeezed into 2 words, or 4 bytes. The task name specifies the protocol, and the node receiving a request message dispatches it to the task designed to serve that particular Acnet protocol. The `srcTaskId` in a reply message received by the requesting node serves to route the reply message back to the requesting task. The `msgId` field allows a requesting task to have multiple active requests outstanding. It is echoed in the Acnet header of the reply message so the requesting task can recognize the request to which the reply refers. The message size includes the size of the Acnet header, so its minimum value is 18, which actually occurs in the case that there is no message body, such as a status-only reply message.

The `pidHi` and `pidLo` together specify an 8-bit property index and a 24-bit device index, which is a reference to an Acnet database entry. The device index is normally not used by front-ends, even though it is included in the protocol. The property index values are few in number. Those most commonly used are:

1	Analog alarm block
3	Basic Control
4	Basic Status
5	Digital alarm block
7	Extended Status
12	Reading
13	Setting

Generic data request support

In the front ends described here, data requests are supported in two steps: request initialization and request updating. During request initialization, the request message is examined for errors, and memory structures are allocated and populated to assist subsequent request updating by making it more efficient. Request updating is performed as indicated by the `ftd` field. Request initialization is analogous to source code compilation that results in object code, which is then executed efficiently as often as needed. Both Classic and Acnet request protocols are supported in this way. The goal is to build efficient support for updating replies so that a large number of active requests can be supported at rates up to 15 Hz.

Network message concatenation

It is possible to concatenate multiple Acnet header-based messages that are destined for the same client port into one datagram, as long as size limits are not exceeded. (UDP datagrams under `vxWorks` are limited to somewhat more than 8K bytes.) System networking support takes advantage of this when it can. A linked list of active data requests is maintained in such an order that all requests from the same requesting node and port (UDP socket) are grouped together. When requests are updated, the linked list is traversed, and multiple reply messages that target the same socket are updated in sequence and queued to the network. When the network queue is flushed, multiple messages that can fit together within the same datagram are easily identified.

Front-end system 15 Hz operation

The organization of the system code operation is synchronous with accelerator operation. Since the original software was designed to support the Linac, each front end operates in a 15 Hz accelerator-synchronous world, driven by a trigger derived from a Tevatron clock event plus delay. This interrupt serves to “start the ball rolling” each 15 Hz cycle by making the `update` task ready for execution. The first of the two principal jobs performed by the `update` task is to interpret the Data Access Table whose entries comprise a list of instructions that cause the data pool to be updated each cycle. (One of

these entries causes all enabled local applications to be called in sequence. These LAs may perform closed loop logic that refers to the data pool and may even modify the data pool.) Armed with a fresh data pool, the second principal job of the Update task traverses the linked list of active requests, and all replies of any protocol that are due to be delivered on the present cycle are built and queued to the network. Finally, the network queue is flushed, causing datagrams to be built and transmitted. The idea of this approach is to update the data pool each cycle with fresh data, then fulfill all active requests as soon as possible. Updating all requests at the same time increases the chance that multiple replies targeting the same socket will be concatenated into the same datagram, thus somewhat improving network efficiency.

Of the remaining tasks each 15 Hz cycle, the next major task to execute is the Alarms task, which checks all active analog channels for changes in the alarm state (good/bad) and queues any resulting alarm messages to the network. The next major task to execute is the Application task, which executes the current page application. (A suite of page applications is included within each front end that can facilitate debugging and system configuration.) At this point, the system rests for the first time in the 66 ms cycle.

Server task

At 40 ms after the start of the cycle, a delay interrupt causes execution of the Server task, which traverses the linked list of active requests and updates any server requests for which replies are due on the current cycle. A server request is one for which the local node acts to collect from other front ends the device data sought by the requesting node. The time that the Server task runs within the 15 Hz cycle is a deadline by which time all data from other contributing nodes must be delivered to the server node so that the entire composite reply message can be sent to the requesting node. In the case of the Linac control system, almost all devices are sourced through node0600, which is often referred to as the Linac data server node. Acnet consoles and other clients send RETDAT requests to node0600 (because the Acnet database source node for most Linac devices says so) that may include device packets that refer to data in many other front end nodes. (At this time, 25 front-ends support Linac.) Node0600 forwards the request via multicast so that all Linac front ends have a chance to see it. Each one that sees at least one of its own devices included in the request takes it upon itself to schedule reply updates of only its own data; otherwise, it ignores the request. The server node, when it initialized the request, scanned it carefully, so it is well aware of every contributing node from which it can anticipate reply reception.

What does a contributing node do when it receives a forwarded request? When it scans the request, it may also notice that several different nodes are represented. The reason it does not assume the role of a server node for the request is that it was multicast. So it scans the request for its own devices, behaving exactly as if the request included no device packets referring to other nodes. The result is that each contributing node to a server request replies with its own part of the reply data. As the server node receives these partial replies, it knows exactly where the data should be copied into the composite reply buffer, again, because it scanned the original request very carefully. Note that a contributing node can be the server node as well. For a request whose device packets reference both the server node and another node, say, the server forwards the request via multicast addressing, and it is received by both the other node and the server node. Each will notice that it has a part to play in the request and prepares to deliver appropriate partial replies. The other node replies to the server, and the server node replies to itself. In this way, the server node supports the same request as both a server and as a nonserver. It wears two hats.

What if the request specifies only data all of which comes from one other node? In that case, the forwarded request is not multicast, but unicast to the other node. Since it is not multicast, will that mean the other node will act as a server? No, it will not, because it will see that all device packets in the request refer to itself. It will merely schedule a reply message that it will return to the server node, which will then forward this complete reply to the original requesting node. One more case. What if all device packets in the request received by the server node are local to the server node? In this case, the

receiving node will not act as a server node. It will merely reply with its own data to the original requesting node.

Front-end performance

The above description of activity scheduled for each 15 Hz cycle may leave the reader wondering if there is time to do all of these tasks at 15 Hz. There is time, by definition, unless the system has not been configured properly. The Update task for IRMs may typically require about 1–2 ms, including execution of all the local applications; some front ends may require a bit more than that. The Alarms task typically requires 1 ms or less. The current page application may require a similar time. The Server task usually requires less than 1 ms. Most IRMs are idle 90% of the time.

For the PowerPC nodes, there is an additional delay required to execute the Update task because much of the Linac data arrives via arcnet from one or more SRMs, or Smart Rack Monitors. The delay to acquire this data may be 10–15 ms. Even though the PowerPCs are intrinsically faster than the MC68040-based IRMs, the Update task runs considerably longer in the PowerPC. (It just happens that the current PowerPC systems are in Linac, where data is collected indirectly through SRMs.) The PowerPC Alarm task executes in about 1 ms or less, which could be faster but for the slow access time (1 μ s) experienced with the nonvolatile memory that houses the system tables, one of which is heavily accessed during an alarm scan. Even so, the PowerPC systems are idle about 75% of the time. But the faster PowerPC-based systems can execute much more code in the remaining idle time than can the IRMs, if they need to do so.

The server node logic described above may seem to represent a lot of network activity. There are timing measurements that illustrate examples such as a data request that is received by the server node that requests data from 18 other nodes. The request is received, forwarded, contributing node replies received, and the final composite reply sent, all in about 5 ms. Here is such an example from network diagnostics that are included in each front-end:

```

F NETWORK FRAMES 10/25/02 1427
NODE<0600> #RCVD= 237 LIST<0509>
NODE=0000 - SIZE>0000 TIME=0000
SrcN Size T ^Frame HrMn:Sc-Cy+ms
A92E 03D8 R E6C882 1426:55-08+19 Received RETDAT request from cns46
A9FC 03D8 T E81EB8 1426:55-08+19 Forwarded request
A600 03D8 R E6CC6E 1426:55-08+19 Received forwarded request
A614 001E R E6D05A 1426:55-08+21 First reply
A61C 0016 R E6D08A 1426:55-08+22
A612 001A R E6D0B2 1426:55-08+22
A613 001E R E6D0DE 1426:55-08+22
A611 001E R E6D10E 1426:55-08+22
A615 0046 R E6D13E 1426:55-08+22
A622 001A R E6D196 1426:55-08+22
A624 001A R E6D1C2 1426:55-08+23
A627 001A R E6D1EE 1426:55-08+23
A625 001A R E6D21A 1426:55-08+23
A626 001A R E6D246 1426:55-08+23
A620 0022 R E6D272 1426:55-08+23
A621 001A R E6D2A6 1426:55-08+23
A623 001A R E6D2D2 1426:55-08+23
A62E 003A R E6D2FE 1426:55-08+23
A62F 005E R E6D34A 1426:55-08+23
A62D 001A R E6D3BA 1426:55-08+23
A61E 002A R E6D3E6 1426:55-08+23 Last reply
A92E 0102 T E822A2 1426:55-08+24 Deliver composite reply to cns46

```

Each line represents the receipt or transmission of one datagram. The times are shown down to units of accelerator cycles (range 00–14) and ms within the current cycle. The server node received the original request at 19 ms, and it delivered the reply at 24 ms. In this example, 18 nodes contributed replies to this one-shot RETDAT request for 60 Acnet devices. Note that 11 of the replies were processed within 1 ms. At this moment, as this note is being drafted, node0600 is supporting a total of 24 RETDAT requests from 11 different requesting nodes. Its CPU utilization is approximately 10%.

The ssdn field of the Acnet device packet

The Subsystem Device Number uses a format that is designed by the front-end programmer and carries no intrinsic meaning to clients. For the front-ends described here, it has the following format:

<i>Field</i>	<i>Size</i>	<i>Meaning</i>
ltypFlgs	2	Listype# in hi byte, flags in low byte
node	2	Ident node# of device data
index	2	Ident index, such as an analog channel#
size	2	Size of one item in low byte, for support of array access

The node# in the second word of the ssdn format allows a server node to easily decide how the request should be handled. Actually, the term “server node” is a bit misleading, as any front-end acts as a server or not depending on what it finds in the node fields of the device packets in the request, and also how the request was received, whether it was multicast or not.

The ltypFlgs field consists of 3 fields. The upper byte specifies the listype# that roughly specifies the type of data that is sought. The nibble in bits 7–4 specifies how the offset word is used. The nibble in bits 3–0 can have a value of 1 or 2, specifying whether the index occupies one word as listed above, or whether the index occupies 2 words, where a common example is a 32-bit memory address. In the latter case, there is no size specification possible. In the former case, the size specification in the low byte of the 4th word is used to support array data access. A simple example of this occurs when accessing an analog channel reading but requesting 16 bytes of data. Without a size specified (size field = 0), one will get a reply consisting of 16 bytes of memory beginning with the targeted 16-bit reading. With a size value of 2, though, the reply data consists of the 2-byte reading followed by 2-byte readings of the next 7 channels.

Listypes and idents

The two made-up words “listype” and “ident” together characterize all the underlying front-end data that can be accessed via a data request, whether Classic or Acnet protocol. Listype numbers currently range from 0–96. Here are a few common examples:

0	analog reading
1	analog setting
2	analog nominal value
5	analog associated status
29	16-bit memory words
50	data stream records

Each listype# uses a specific ident type. The ident type used by listypes 0, 1, 2, and 5 consists of a node# and a 16-bit analog channel number. The ident type used by listype 29 consists of a node# and a 32-bit memory address. The ident type used by listype 50 uses a node# and a data stream table index number. An ident can only be interpreted in the context of a listype# that specifies its type. Each ident type has its own fixed length. (An extreme case is that of a named program file ident, whose length is 14 bytes, including the node#, an 8-character file name, and a 4-byte offset.) Listype 2 is normally used by Acnet to access the Analog Alarm Block property.

Length and offset

The last two words of an Acnet RETDAT device packet are the `length` and `offset` words. The `length` is simply the number of bytes of data requested. The `offset` can have more than one meaning, according to the `ltypFlgs` field in the `ssdn`. If the offset specification in bits 7–4 is zero, then the `offset` word cannot be nonzero, with the exception of a reference to a digitized waveform, such as those used by Booster BLM front-ends, in which it is used as a simple byte offset into the waveform array. (More on this is described later.) If the offset specification is 1, a nonzero `offset` is internally applied to the `index` value. If the `index` value is zero, for example, and the `offset` is `0x0100`, then with a suitable `listype#`, analog channel `0x0100` would be accessed. But the result would be the same if the `offset` word were zero and the `index` value were `0x0100`, so why do it? The reason to support this feature is that the `ssdn` is fixed in the Acnet database; the application programmer cannot modify it before sending a request to a front-end. But both the `length` and `offset` words are specifiable by the requesting program. This feature allows access to any of many possible devices within a front end by defining only a single generic device in the Acnet database. It is most often used for system management. Finally, an offset specification of 2 is a special case that permits use of the 16-bit `offset` multiplied by 256 before being added to a 32-bit `index` value. It would allow a single Acnet device to reach anywhere within 16 MB of memory, for example. (It is quite possible that this feature has never been used.)

Enhanced RETDAT support

The generic RETDAT support described above, in principle, covers most Acnet needs for device data access in these front ends. But over the years, enhancements have been made to satisfy particular needs by users. The first special need was to support beam-centered averaging, because Acnet clients preferred not to request Linac data replies to be returned at 15 Hz. Other parts of the accelerator normally access data at 1 Hz, and it was felt that Linac should not be treated any differently. This was a problem, because in the Linac, every 15 Hz cycle is a “new ball game,” in which beam may be accelerated, or not, according to the current Time Line Generator configuration. A request specifying 15 Hz replies, usually used by Acnet, would not have a good chance of observing Linac beam cycles. Long ago, before Acnet was designed, Linac data was collected at 15 Hz, with beam-centered averaging logic applied by the client program, such as the generic parameter page. Beam-centered averaging is an algorithm that averages data collected at 15 Hz, with priority placed on beam cycles. In the absence of beam cycles over the averaging period, the simple average of all cycles is used. Consider a case where data is sampled from 15 cycles, and beam was scheduled on the 5th and 12th cycles only. The result of beam-centered averaging is the average of the readings sampled on the 5th and 12th cycles, with the readings on the other cycles ignored. In this way, the data collected at 1 Hz exhibits beam data, or in the absence of any beam cycles, a simple average of readings on 15 consecutive (non-beam) cycles. The client does not miss beam data because of only sampling at 1 Hz, where, without this support, data from 14 cycles out of 15 would be ignored.

A second special feature added to RETDAT support allowed for flexible support of waveform access. Booster BLM data consisted of sampled readings of loss monitors, but they also needed to access the waveforms of these signals sampled by a digitizer at 12.5 KHz, providing 500 data points over 40 ms, which includes the entire Booster acceleration cycle. The users needed to access both the single values and the waveforms via RETDAT, but they did not want to have separate device names to cover the two cases. There is only one property that can be used for this, the `reading` property, so using different properties was not feasible. Special logic added to RETDAT took notice of the `length` and `offset` values used to determine what was sought. If the `length`=2 and the `offset`=0, that meant that the single reading, say the final BLM integrator signal, would be accessed. If the `length`=2, but the `offset` is nonzero, it would allow access to any point in the waveform—except the first one. If the `length` > 2, it would allow access to any subset of the waveform. This could all be done using only one name (and one `ssdn`) per BLM.

But that was not all. It was desired to collect BLM waveforms at 15 Hz. But Acnet does not have client support for reliable 15 Hz data collection. The reason is that Acnet client applications run

asynchronous with the accelerator. They use the operating system to schedule execution at intervals of 60, 70, and 70 ms, resulting in an average rate of 15 Hz, but one that slips in phase with the accelerator 15 Hz, since the latter is synchronized with the power line frequency, which varies somewhat during the day, whereas the former is presumably driven by a crystal, whose frequency does not vary. With an application access to data based upon a data pool, the result is that it is easy to imagine two successive 15 Hz replies from a front end occurring between two consecutive executions of the application, so that one reply is missed. After consideration of the situation, it was decided to provide support for supplying two sets of data, each set captured from two consecutive 15 Hz cycles, at 7.5 Hz. The application would need to expect this and separate the two sets of data for processing. In order to make it possible to correlate data from multiple front ends, the returned structure includes a cycle counter whose value is equal in all front ends. The application would have to watch for the reply data at 15 Hz as best it can, but it will only find new reply data, on the average, every other time it checks. It is extra work for the application code, but it was the only solution found that would solve the problem of reliable access to 15 Hz accelerator data.

The solution described here was called time-stamped data, where the time stamp is merely a cycle counter. In order to recognize a RETDAT request for such data, the unusual reply frequency of 7.5 Hz is the key. If the `length` is large enough, and the requested reply rate is 7.5 Hz, then a structure that includes a header and two sets of data is accumulated in the front end on successive cycles for delivery to the requesting node. The sets of data can be waveforms or other data. The `length` includes 4 bytes for a header and twice the size of the set of data sought. All of these options are supported via the additional features added to RETDAT.

Why not FTPMAN?

After all this discussion of the time-stamped support for RETDAT to gain 15 Hz access to waveform data, one may ask, why not use FTPMAN to access waveform data? The answer is that FTPMAN cannot support access to 15 Hz data, unless one considers its continuous option. (The snapshot option is suitable for capturing an isolated waveform that was collected at some sample rate following a clock event plus delay, but it cannot keep up if such events occur at 15 Hz.) But the continuous option is currently limited to 1000 Hz, or perhaps 1440 Hz, whereas the waveform data used for BLMs is sampled at 12500 Hz. As possible future support, one may ask, what about support for HRM data? The new HotLink Rack Monitor samples data continuously at 10000 Hz (100 μ s per sample), so the same limitations apply, and FTPMAN is not suitable for HRM waveforms at its full rate, either. One could sample HRM data at, say, 1000 Hz using the FTPMAN continuous option, but this would likely be considered inadequate for measuring Booster acceleration cycle waveforms that only last 33 ms; one would collect only 33 points at most per cycle.

One may ask, why not collect data via RETDAT on an event that may, but often does not, occur at rates anywhere close to 15 Hz? If one has events occurring in pairs of 15 Hz cycles that occur every 3 seconds, for example, it would seem that the time-stamped approach described above is overkill; nearly all of the 15 Hz waveform data would be thrown away. Using RETDAT and specifying a clock event for replies would produce pairs of replies every 3 seconds, which would often be successful, but not always, for the same reason that RETDAT cannot reliably collect 15 Hz replies when they occur less than 70 ms or so apart.

What about queuing? If RETDAT access, which relies on a client data pool system that may be updated by 15 Hz replies every cycle, has difficulty delivering 15 Hz data to an application program, why not queue the data, so the application can find it even if it is slightly late? In other words, why not use queuing rather than pooling? This approach is possible using queuing in the front end. A mechanism for this support is provided by "data streams," which refers to a formalized circular buffer scheme in which multiple clients can access data stream records without interference. A local application can record data sampled at 15 Hz into data stream records, and clients can access such data stream records, the support for which maintains a pointer to the last record delivered with the request support structure, so that the next reply carries on from that point. There may be limits in how many waveforms can be

accessed in this way, but it does allow reliable delivery of 15 Hz data, in principle.

What about Java, on which the emerging Fermilab control system relies? Although the client support is still under development, it does support call-backs, in which a client method is invoked when fresh data arrives from front ends. This scheme shows promise, as there is no artificial delay of up to 70 ms before the application has a chance to sample a data pool. Even if it only provides prompt notification of new data that is available for sampling from a data pool, it should be good enough, unless occasional delays caused by Java garbage collection get in the way. Java support promises to eventually provide access to correlated 15 Hz data collected from different front-ends.

Summary

This note briefly described the RETDAT data request protocol and how it is supported in the IRM and Linac-style PowerPC front ends. The support for the protocol is embedded into the 15 Hz operational organization of the system software that furnishes the underlying data pool to which the protocol refers. The front end performance was illustrated to indicate how much idle time is available for additional loads beyond the typical 15 Hz activity. Some special enhancements to the RETDAT support, made in response to user needs, are described. These include beam-centered averaging, waveform access, and time-stamped data support, for reliable access to 15 Hz data. Finally, there are some comments about the general problem of access to 15 Hz data in the Acnet environment.